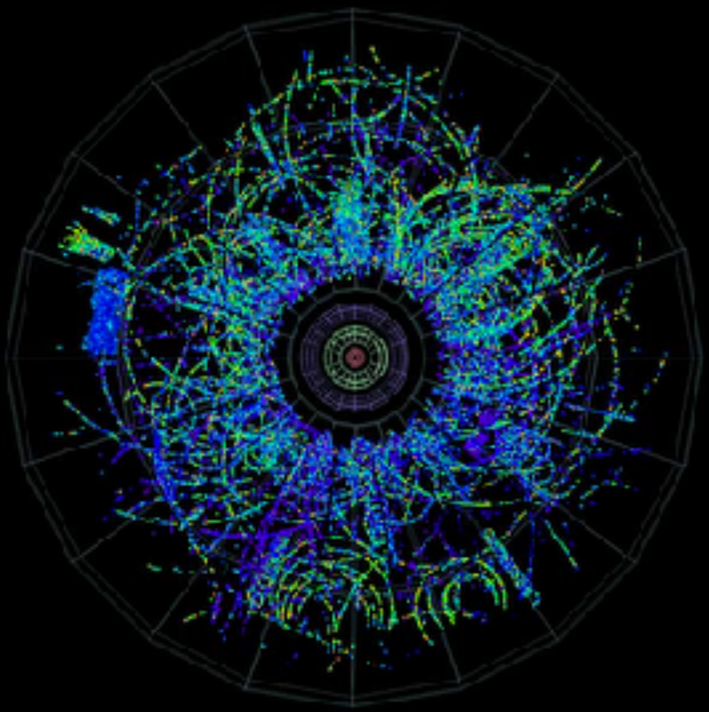
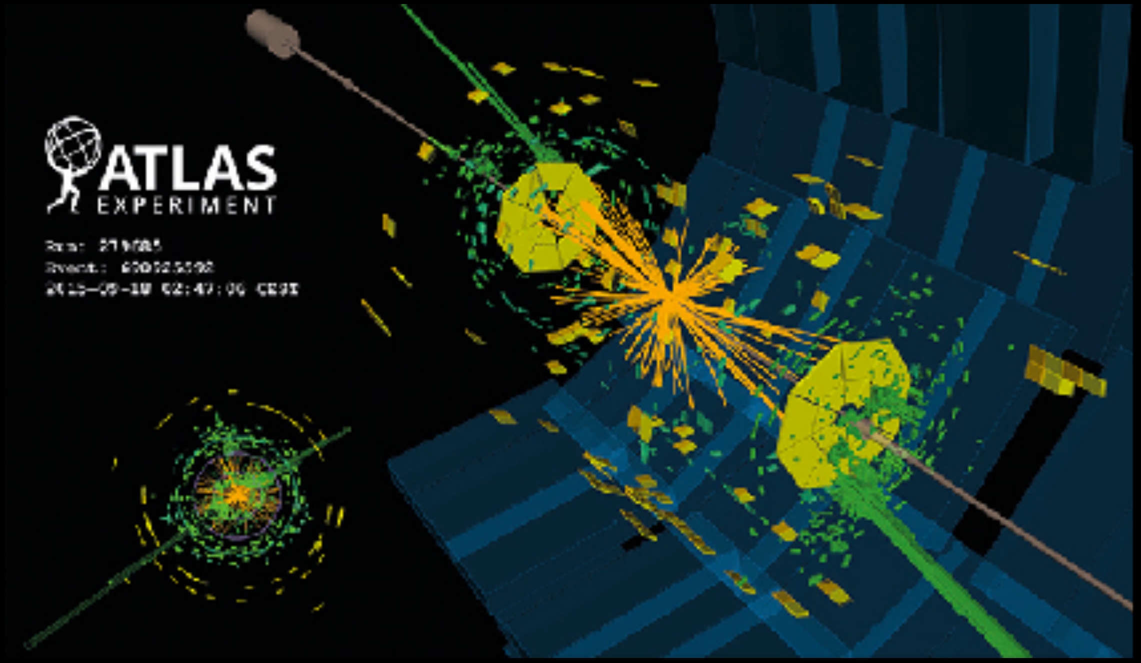
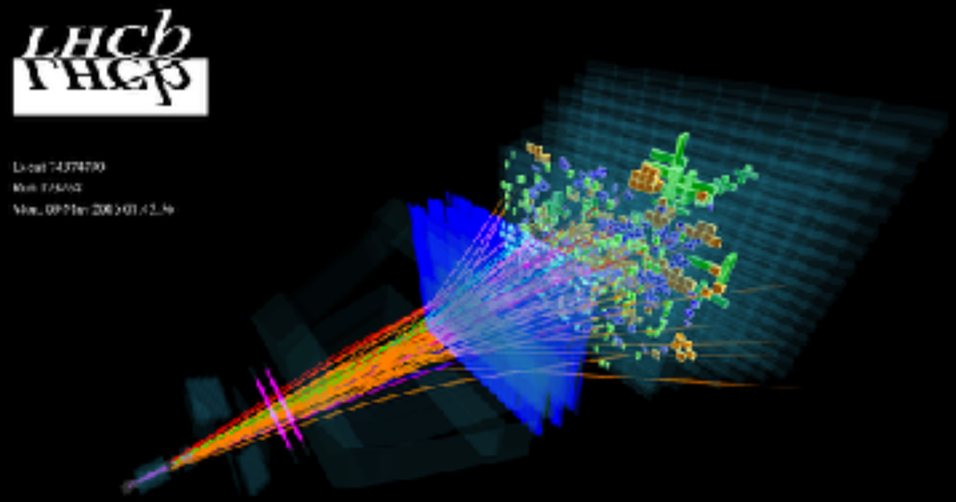


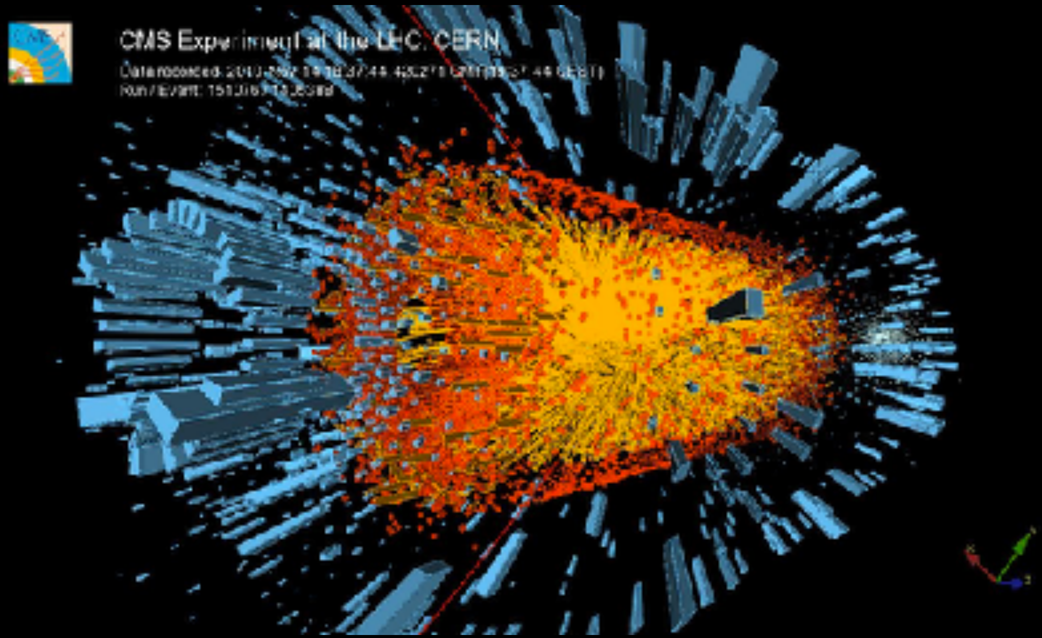


Drones: Making faster and smarter decisions with software triggers

Sean Benson, Konstantin Gizdov

IML Workshop 11/04/2018





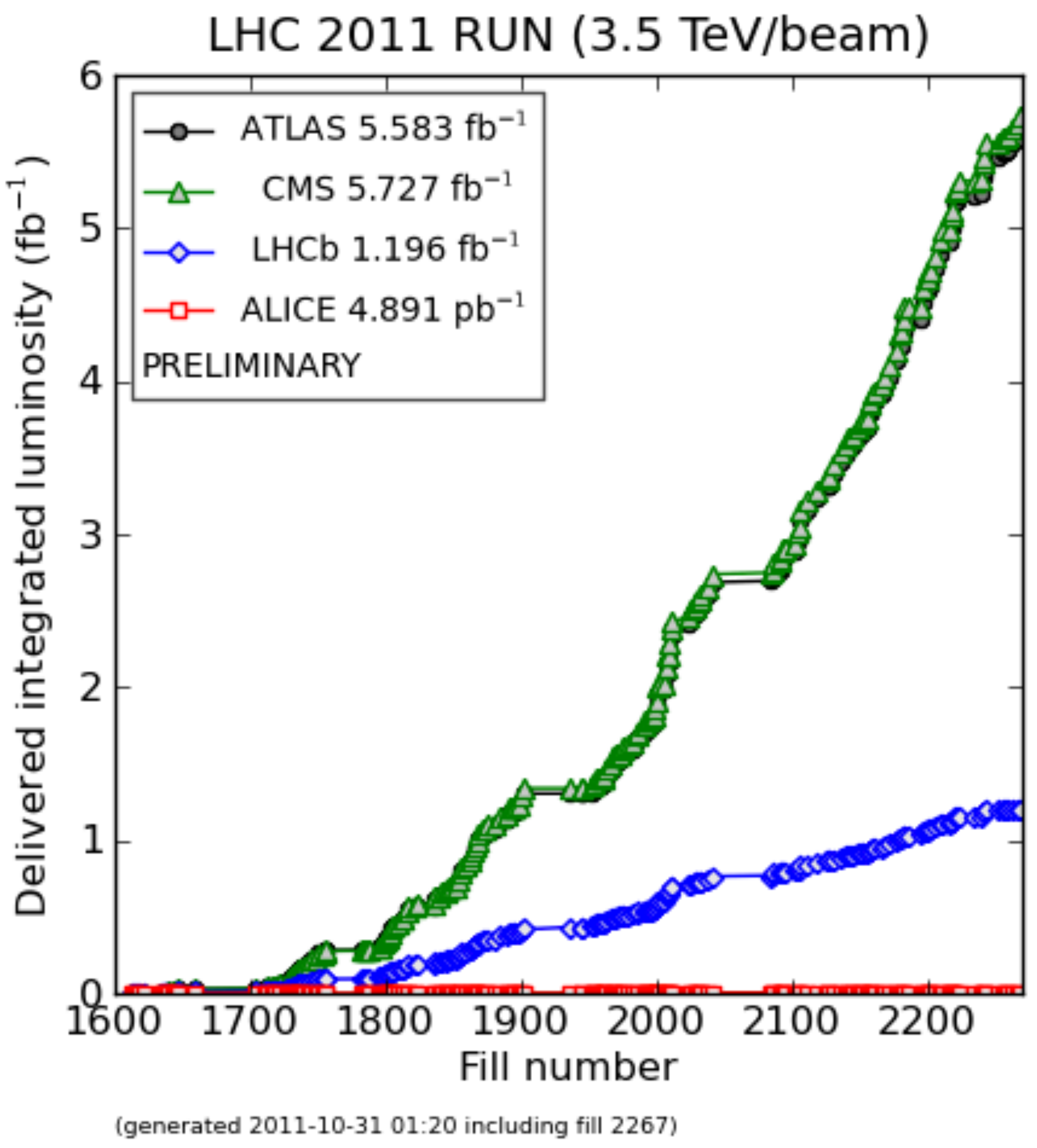
Introduction: The size of the problem/opportunity

$1 fb^{-1} \sim 10^{12}$
proton-proton collisions

each bunch crossing produces:

- 1.5-2.0MB @ ATLAS/CMS
- 60kB @ LHCb

(8MB @ ALICE leading to much lower instantaneous lumi.)



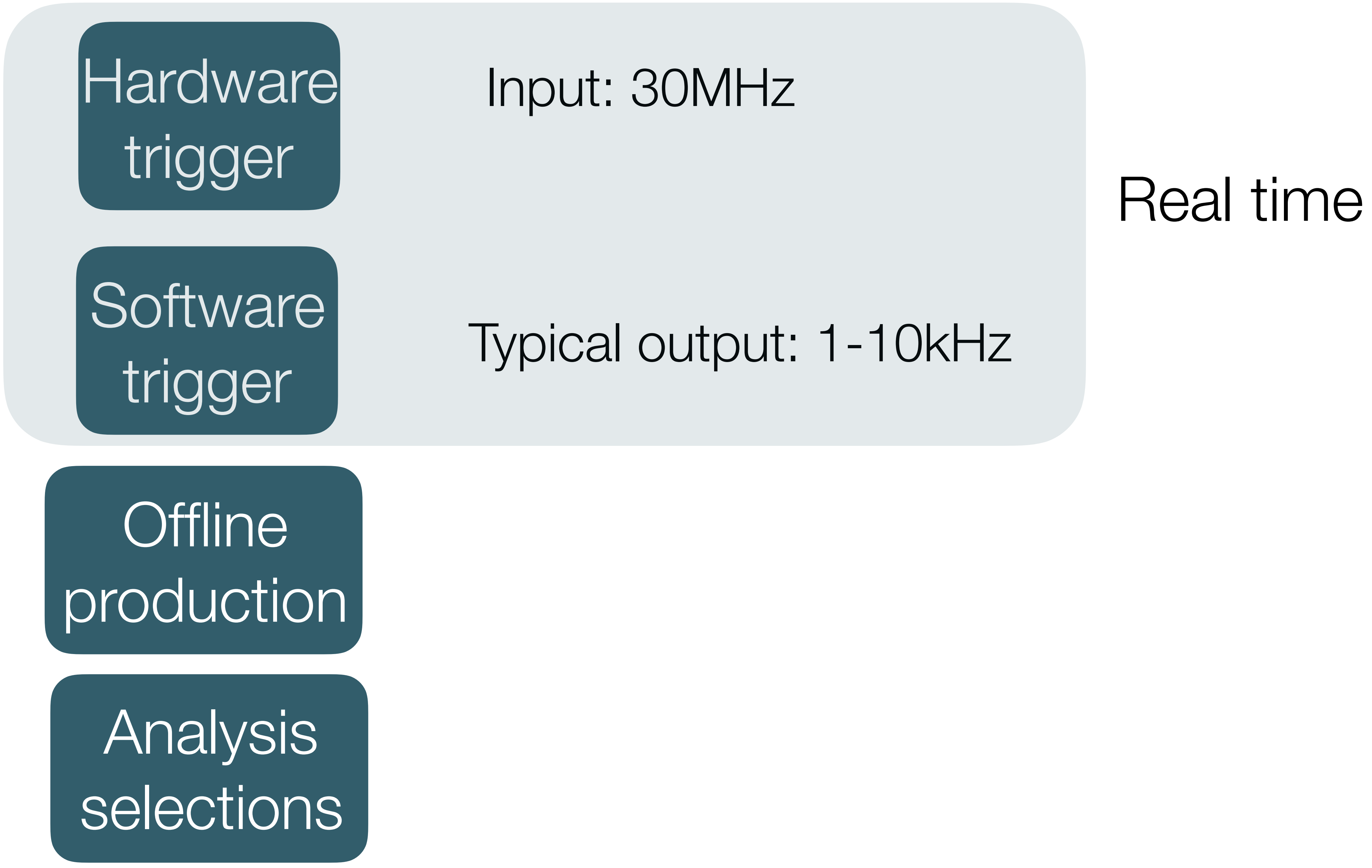
Introduction: LHC data acquisition

Wide range of physics goals across experiments.
However, there is a common approach to data collection:

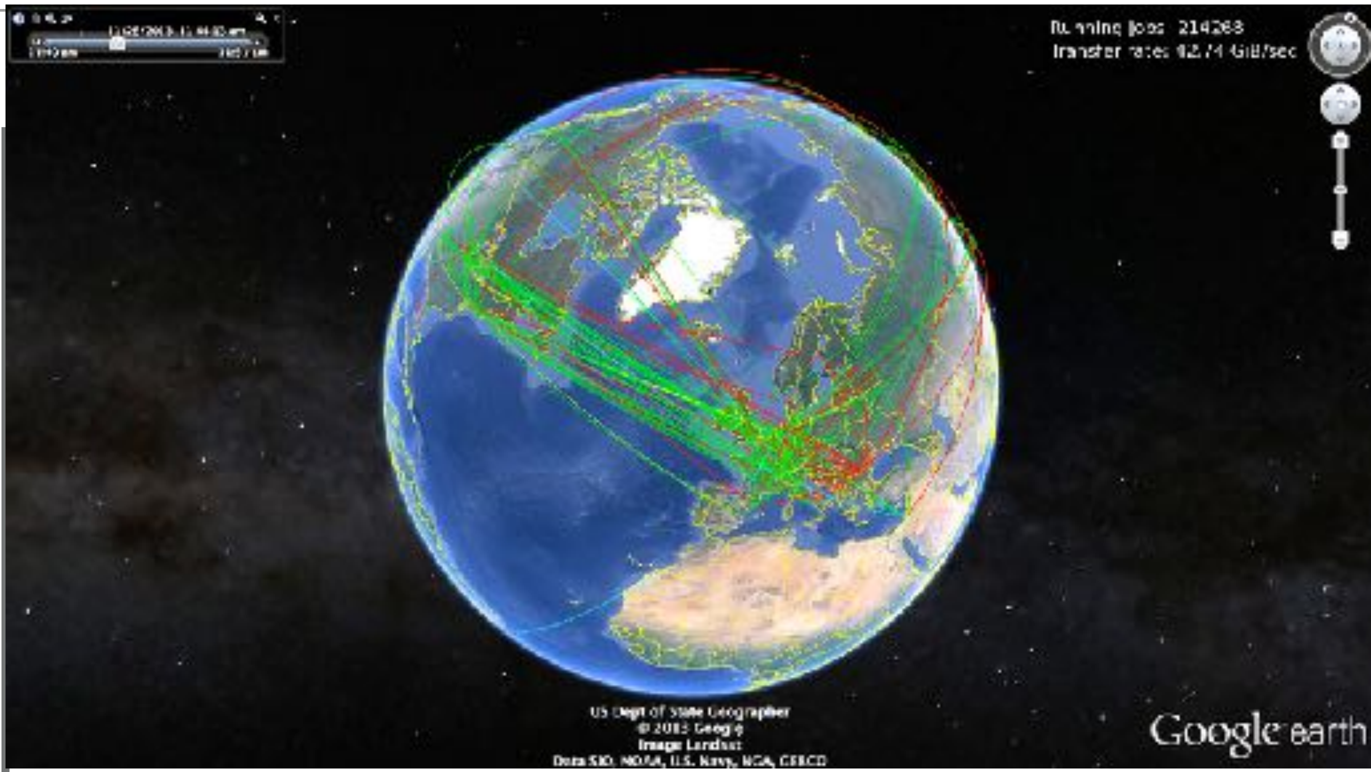
Hardware trigger	FPGA-based determination of quick-to-access information, such as calorimeter energy
Software trigger	More detailed information, from a more advanced reconstruction, track finding, particle ID, etc.
Offline production	Further centralised data reduction using full reconstruction and final calibrations
Analysis selections	Final data reduction using our favourite software tools.

Introduction: LHC data acquisition

Wide range of physics goals across experiments.
However, there is a common approach to data collection:



Introduction: Computing for the LHC

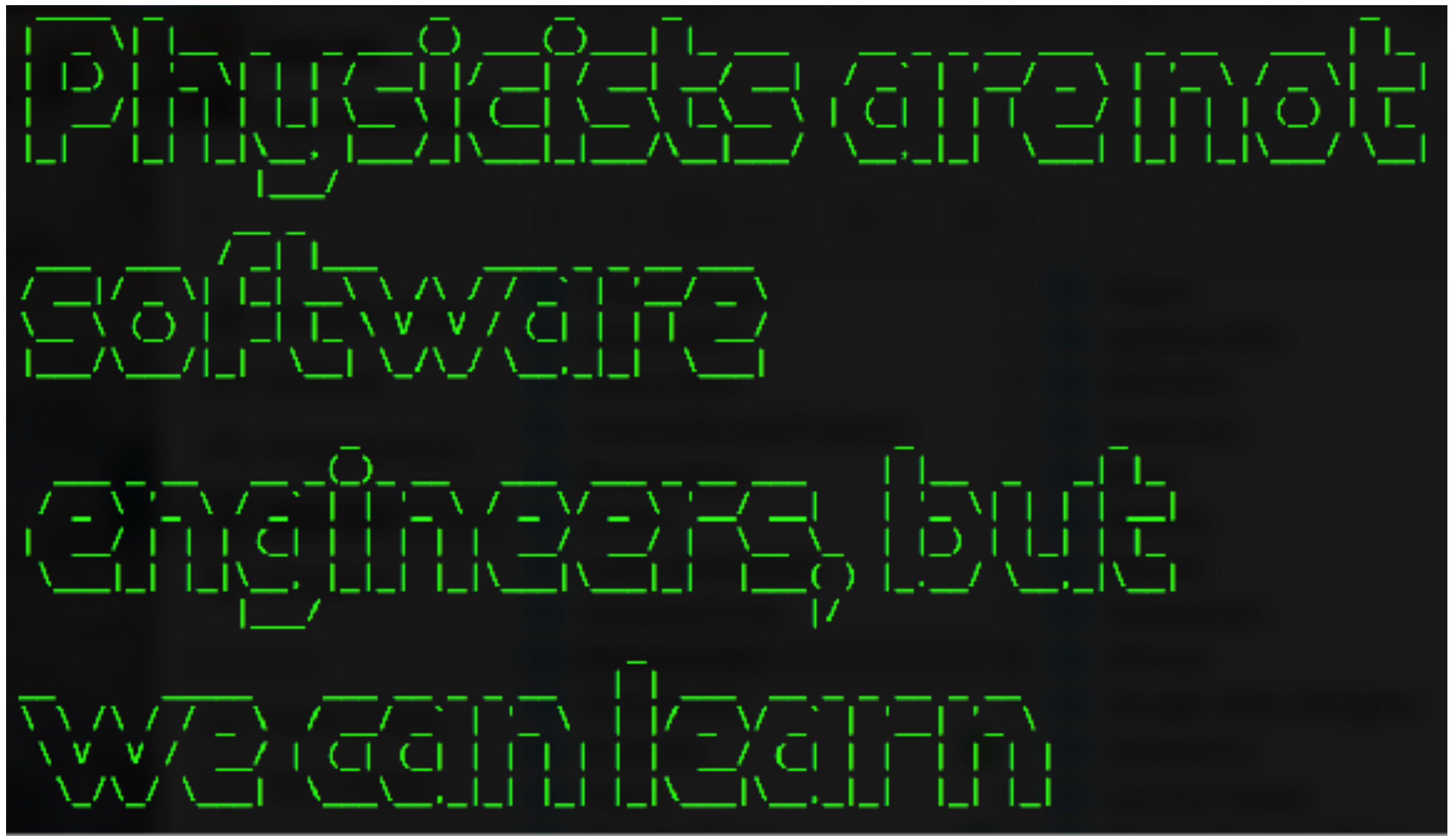


Online: Event filter farms

- Located close to each experiment
- Mini data-centres consisting of 1000s cores

Offline: WLCG

- 42 countries
- 170 computing centres
- 2 million tasks run every day
- 750,000 computer cores
- 400 petabytes on disk and 400 petabytes on tape



Physicists are not software engineers, but we can learn

Tangent: Standard HEP experimental physicist mindset

Introduction: Standard HEP experimental physicist mindset

Hardware trigger

make really basic cuts

understandable basic info.

Introduction: Standard HEP experimental physicist mindset

Hardware trigger

Software trigger

make loose cuts

still do this even though we have almost all the info we want.
-> understandable, we need to be fast

Introduction: Standard HEP experimental physicist mindset

Hardware trigger

Software trigger

Offline data reduction

drag our heels and
be as loose as possible
(mainly with cuts)

Introduction: Standard HEP experimental physicist mindset

Hardware trigger

Software trigger

Offline data reduction

Analysis sel.

Now do something smart with our favourite tool

Overall point

Analysis sel.

When we get to this stage, we often know what was the best we could have done

What was the point of that aside?

There is logic behind such a mindset
 - if I keep my options open as long as possible, then I can make sure I do the best job possible.

Using final analysis packages earlier requires that they are supported in a specialised "production environment" and using dedicated computing farms at the detectors

-> If a new package made, needs to be brought into software stacks.

But I can already use TMVA in production environments!
True, but 100s of TMVA ANNs running next to each other is not fast.

Software triggers

But such production environments are highly specialised because they need to be.

Requires well behaved code/models that can perform reconstruction and selections in a time constrained environment.
-> Software triggers need to perform orders of magnitude data reduction in close to real time.

2 main challenges (summed up in Luke’s talk on Monday):

- Ensuring the model can be evaluated with low latency
- Supporting the technology of tomorrow (serving layer).

MV methods used now: BBDT - low latency

Efficient, reliable and fast high-level triggering using a bonsai boosted decision tree - Gligorov & Williams, JINST 8 (2013) P02013

Central idea: discretise the input features used to create the BDT

Computing result: Evaluation means consulting look-up table
- If/else statements in BDT evaluation -> 1D array

Physics result: Similar performance to normal BDT

Evaluated on toy model separating B from D decays using p_T and IP info.

type	$\varepsilon_{4-\text{body}}$
cuts	63%
BDT	77%
BBDT	74%

MV methods used now: Main BBDT drawback

While it is proven (indeed, a large number of LHCb analyses rely on its use in the software trigger), the method has a significant limitation:

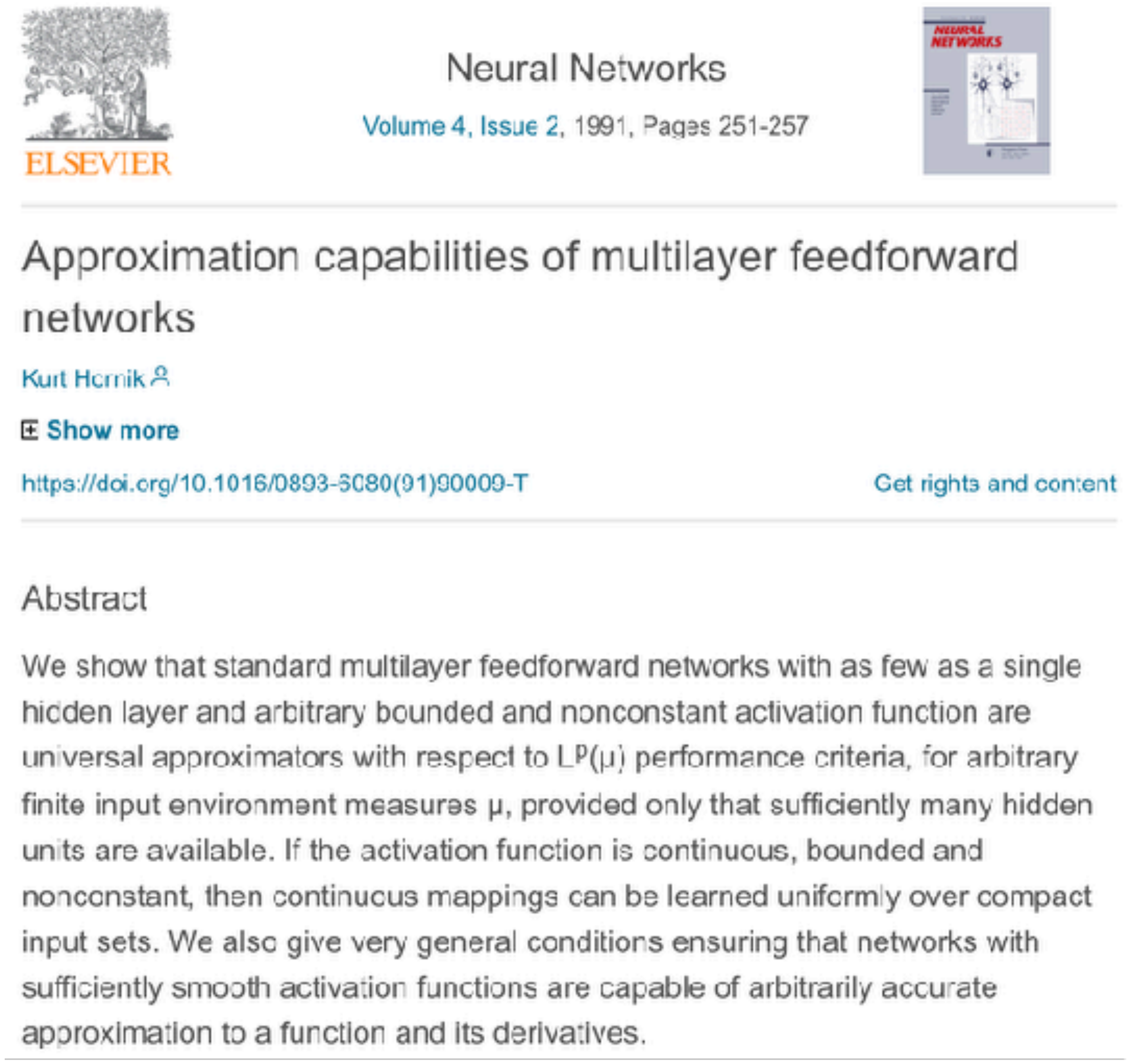
- Does not scale with the number of inputs:
  - Assuming 10 bins / input, new input increases an array size by an order of magnitude.
  - A typical analysis uses 10+ features => quickly run to GB of lookup table per analysis.

Is there another way?

Looking at the problem another way…

All models used for background rejection map inputs to probabilities

If we have a complex multi-layer model, we can approximate it to arbitrary precision using a model that we can make more suitable to low latency environments.



ELSEVIER

Neural Networks

Volume 4, Issue 2, 1991, Pages 251-257

Approximation capabilities of multilayer feedforward networks

Kurt Hornik

Show more

https://doi.org/10.1016/0893-6080(91)90009-T

Get rights and content

Abstract

We show that standard multilayer feedforward networks with as few as a single hidden layer and arbitrary bounded and nonconstant activation function are universal approximators with respect to $L^p(\mu)$ performance criteria, for arbitrary finite input environment measures μ, provided only that sufficiently many hidden units are available. If the activation function is continuous, bounded and nonconstant, then continuous mappings can be learned uniformly over compact input sets. We also give very general conditions ensuring that networks with sufficiently smooth activation functions are capable of arbitrarily accurate approximation to a function and its derivatives.

Construction

trained network

scikit learn

Vanilla network

Small single hidden layer, sigmoid activation

reward drone for giving the same response

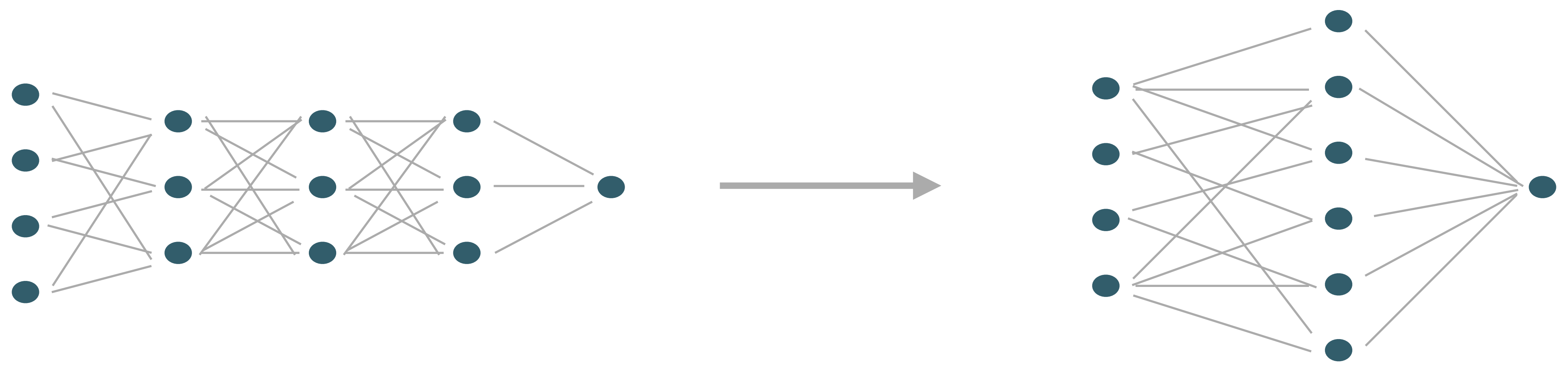
if convergence, add more degrees of freedom to the hidden layer

$$\mathcal{L} = \sum_i (F(\vec{x}_i) - G_i(\vec{x}_i))^2$$

$$\delta \equiv (\mathcal{L}_j - \mathcal{L}_{j-1})/\mathcal{L}_j < \kappa,$$

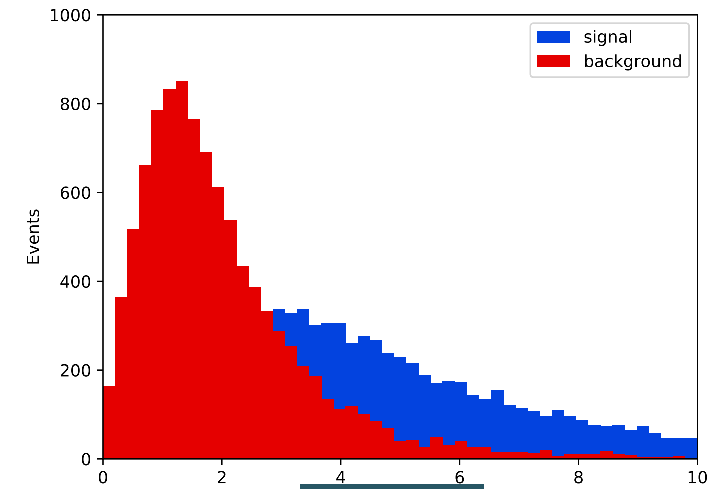
$$\mathcal{L}_j < \hat{\mathcal{L}} - \delta\mathcal{L}_j,$$

update with standard SGD

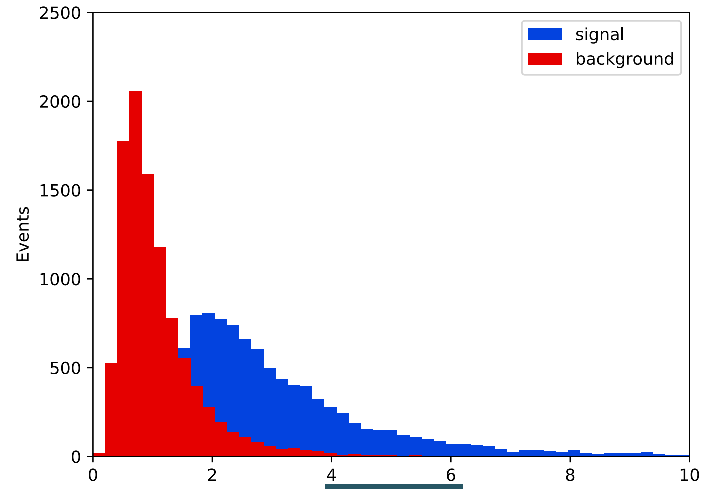


Putting it into action - B decay separation

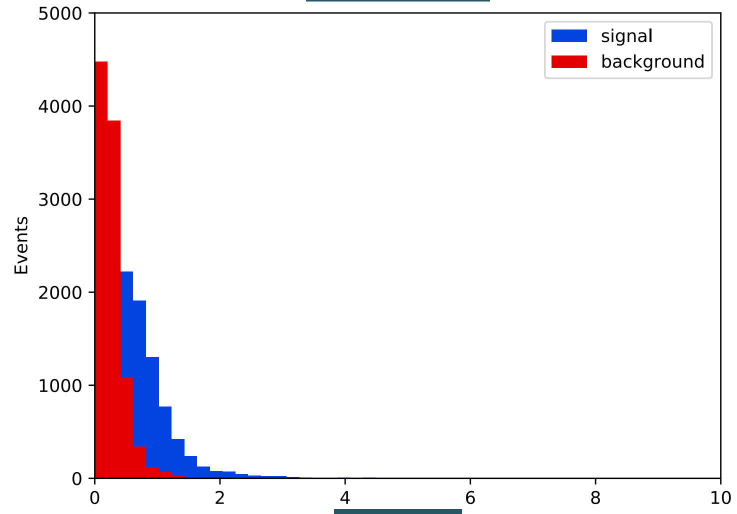
B decay use case -> Separate B->J/ψϕ from D->ππππ



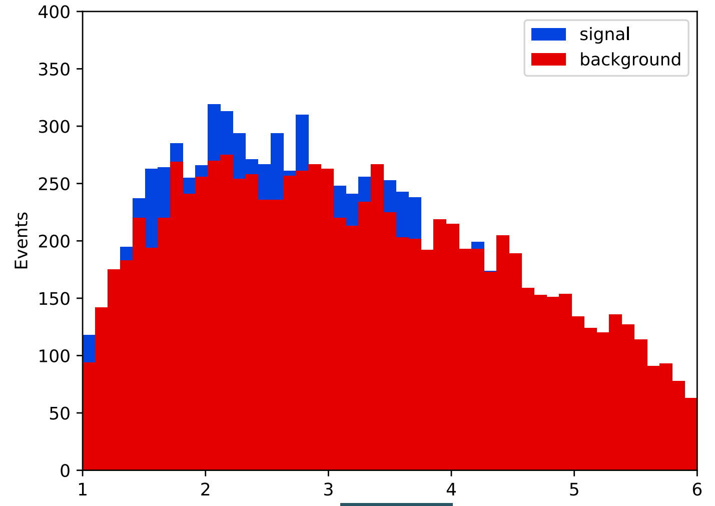
mother p_T



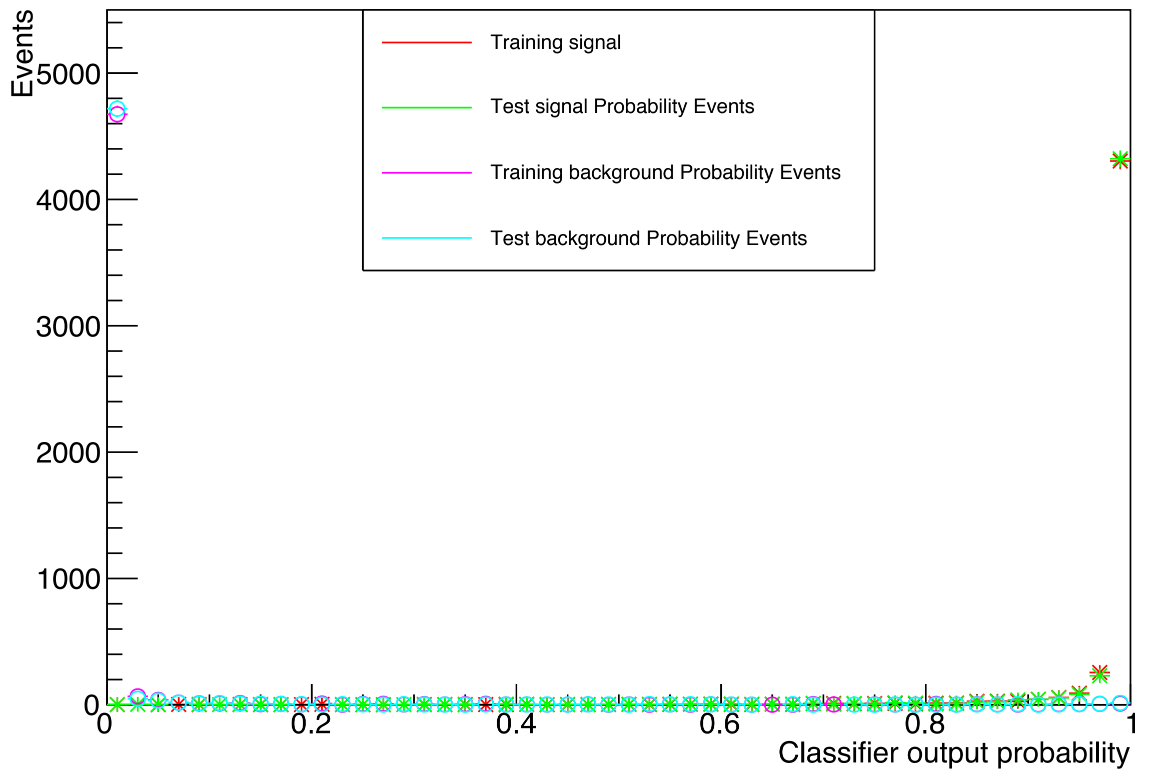
max p_T



min p_T



min η

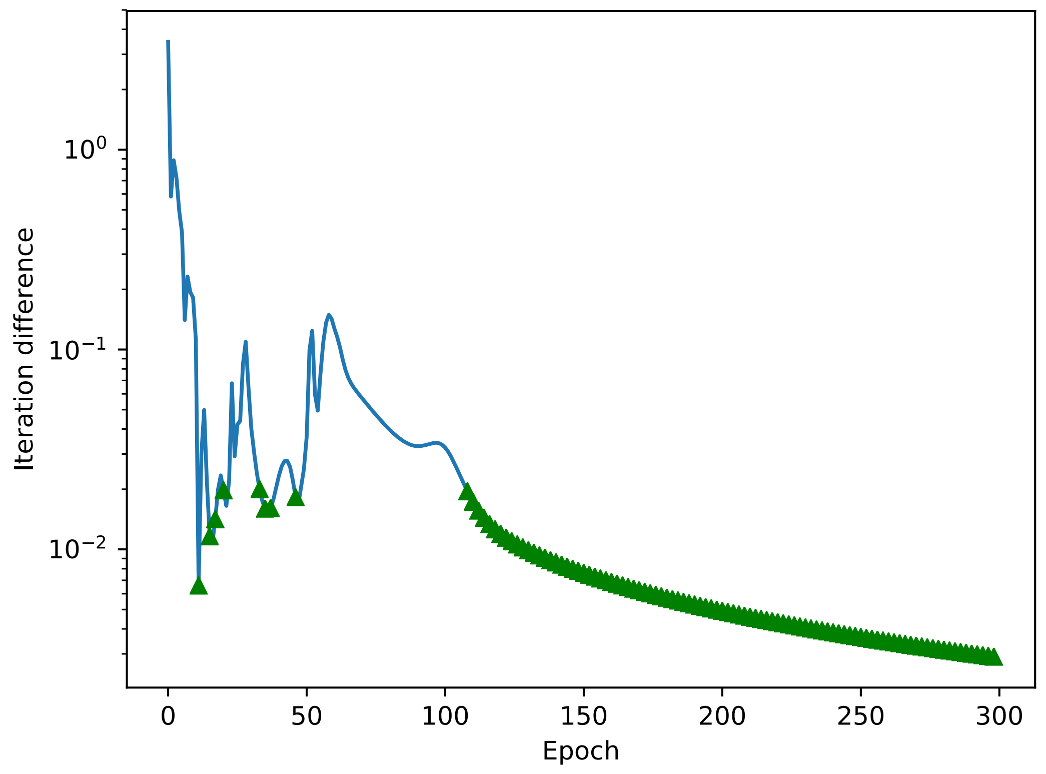


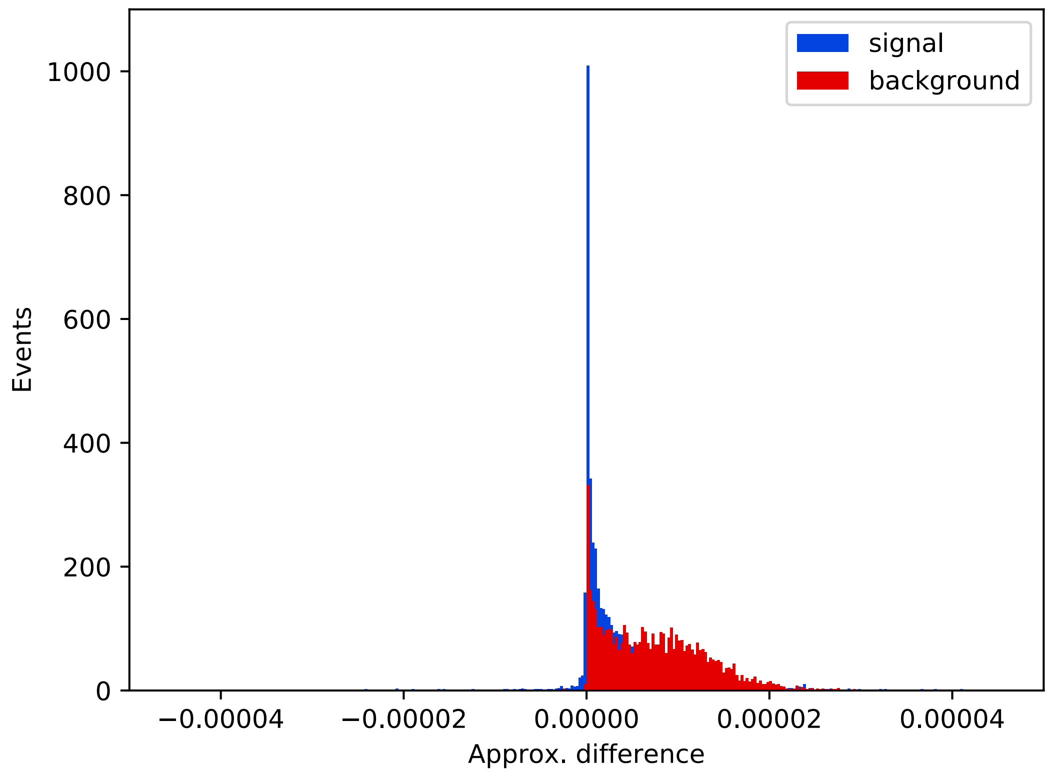
Generate signal and background
from RapidSim - arXiv:1612.07489

train with sk-Learn

Putting it into action - B decay separation

Train drone from sk-Learn classifier



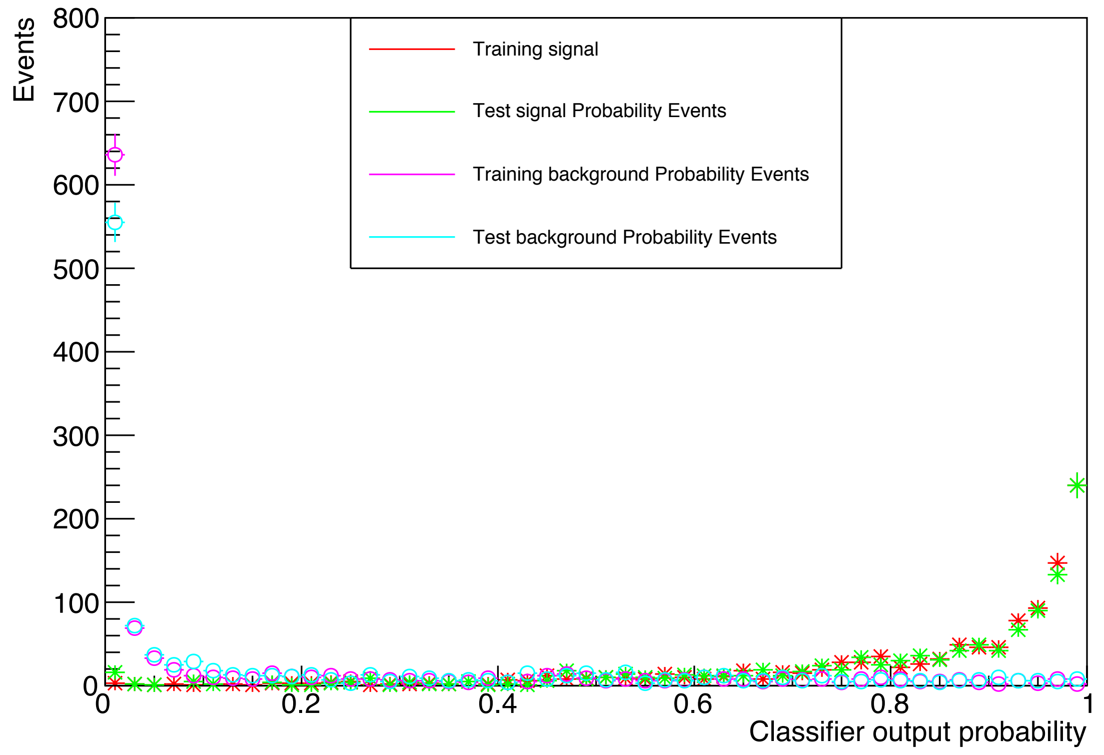


300 epochs
alpha (learning rate): 0.05
layer extension threshold: 0.05

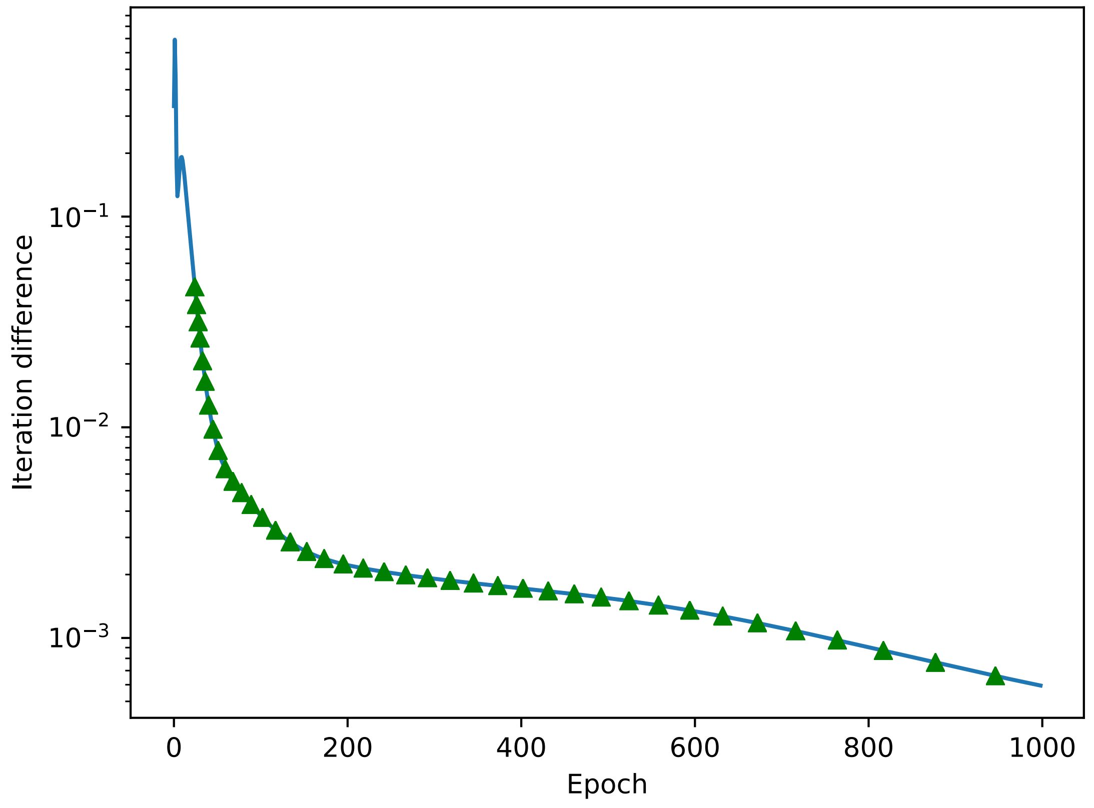
Result is a drone that for each data point, gives the same value as the original

Putting it into action - Jet separation

Generate jets using Pythia + FastJet
Signal: Jets produced in association with W
Background: QCD jets



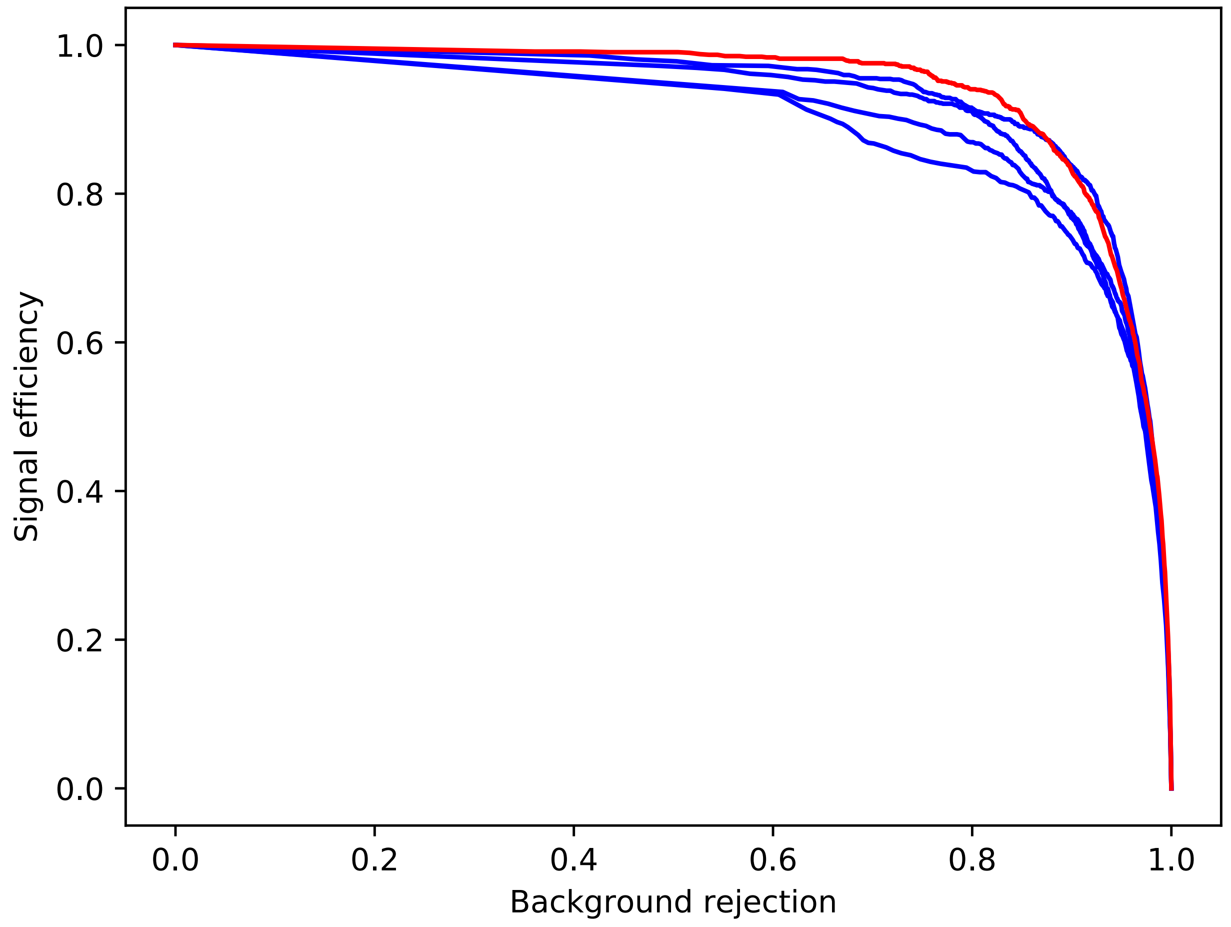
Train sk-Learn MLP:
relu activation, 3 hidden layers



No real change to the drone creation procedure

Performance in detail

True measure of performance is found from the ROC curve: Signal eff. vs. background rejection



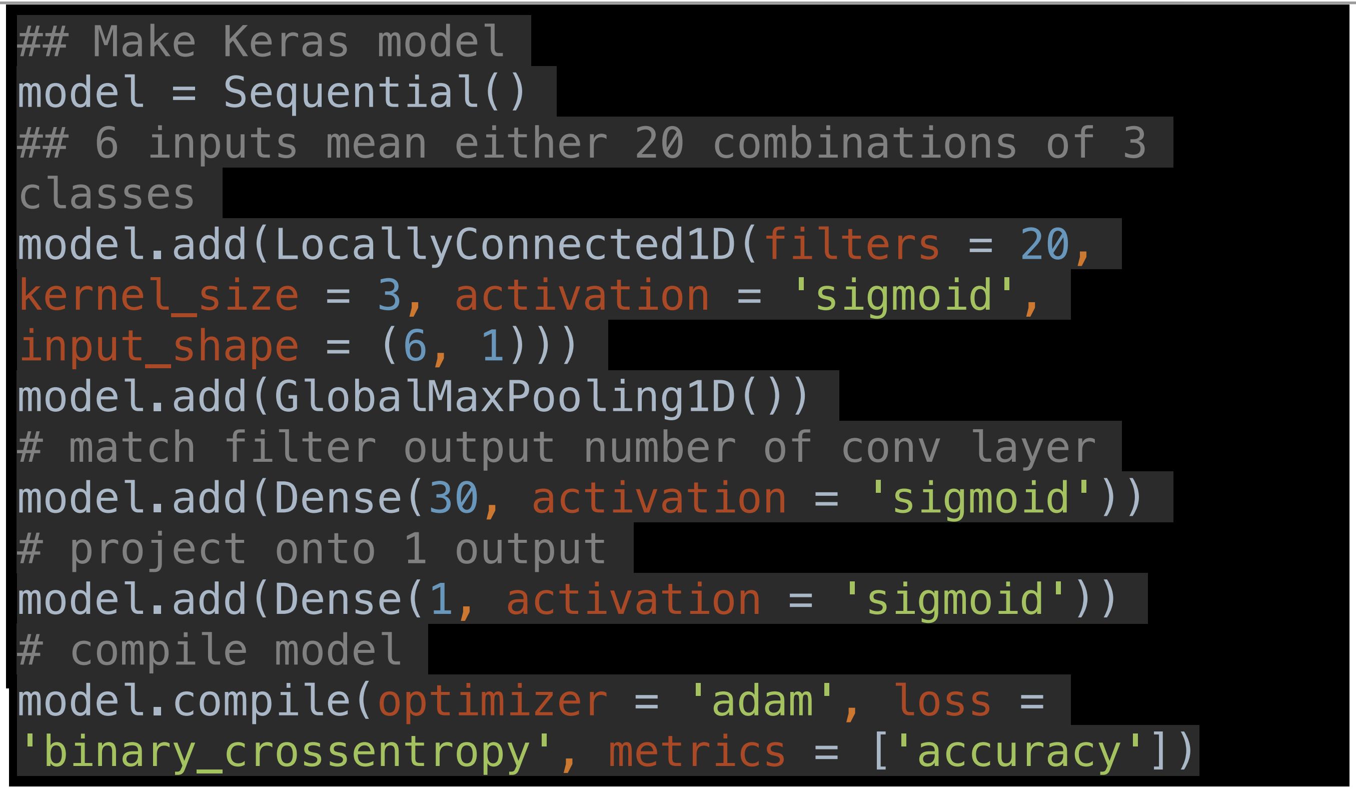
Shown are the ROC curves of the original sk-learn classifier (red) vs:

- drone 100 epochs
- drone 500 epochs
- drone 1000 epochs
- drone 1500 epochs

Conclusion: we can train an approximation network with same performance but simpler structure than the original network

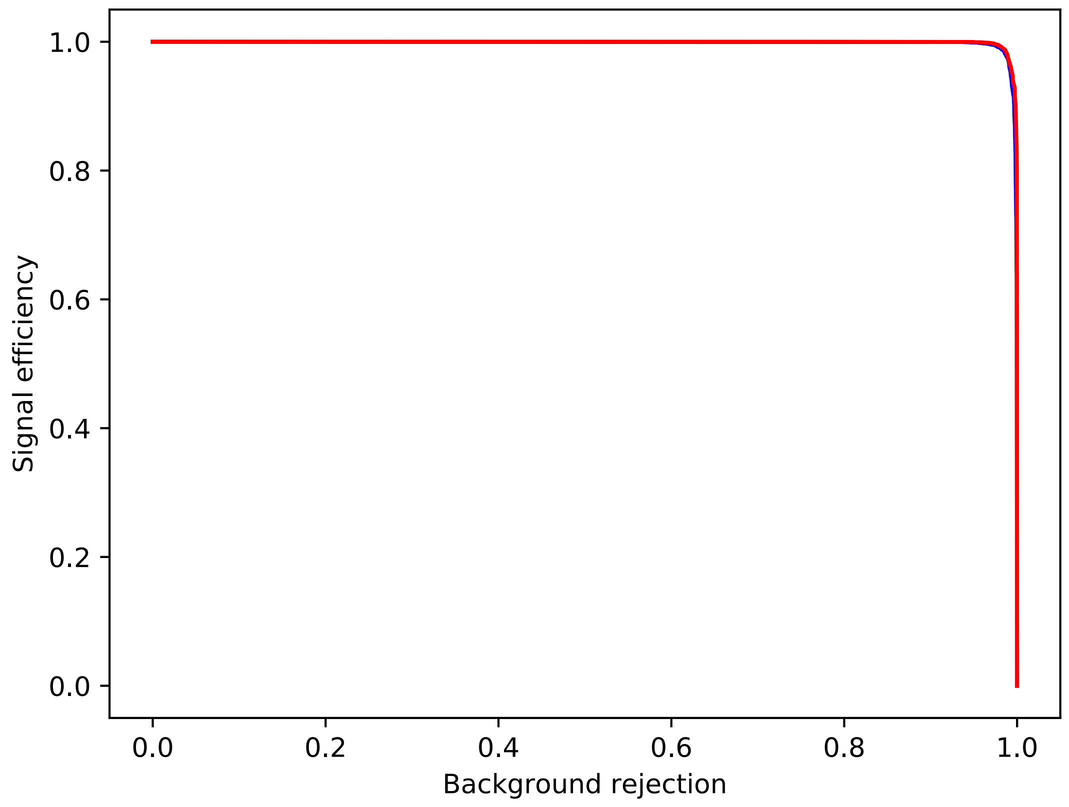
Testing further: Convolutional NN

Take our B and D data before.
Make a more advanced Keras network:



```
## Make Keras model
model = Sequential()
## 6 inputs mean either 20 combinations of 3
classes
model.add(LocallyConnected1D(filters = 20,
kernel_size = 3, activation = 'sigmoid',
input_shape = (6, 1)))
model.add(GlobalMaxPooling1D())
# match filter output number of conv layer
model.add(Dense(30, activation = 'sigmoid'))
# project onto 1 output
model.add(Dense(1, activation = 'sigmoid'))
# compile model
model.compile(optimizer = 'adam', loss =
'binary_crossentropy', metrics = ['accuracy'])
```

Single hidden layer drone, same algorithm as previously described



ROC curves show identical performance

Remarks

Drone never sees any data or labels, only asks the original model what it thought of a point in the parameter space
=> Learns from other models, not from the data

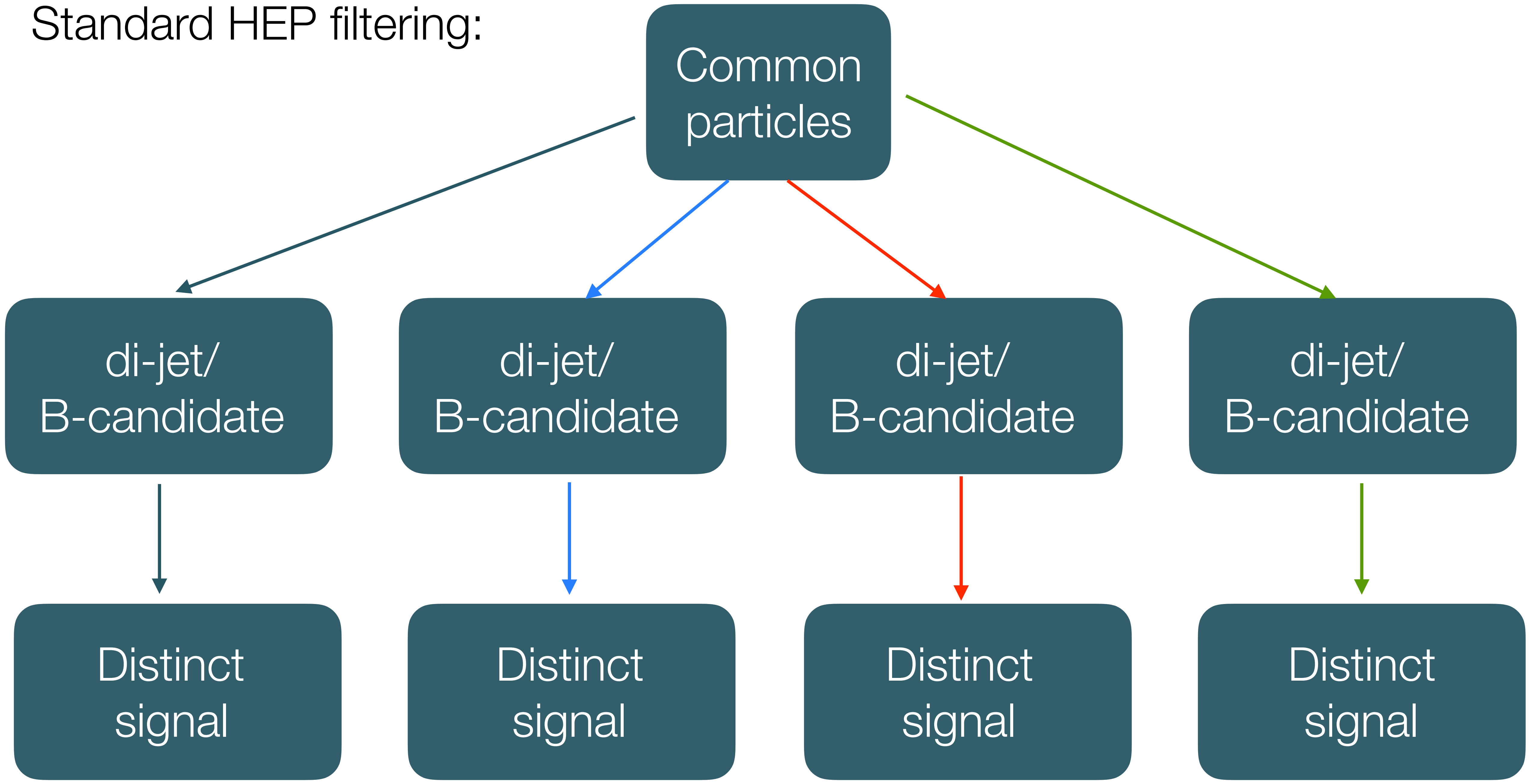
Analysts have complete freedom to design original networks, standardisation means all can be ported to the low latency environment

No need to re-invent the wheel:

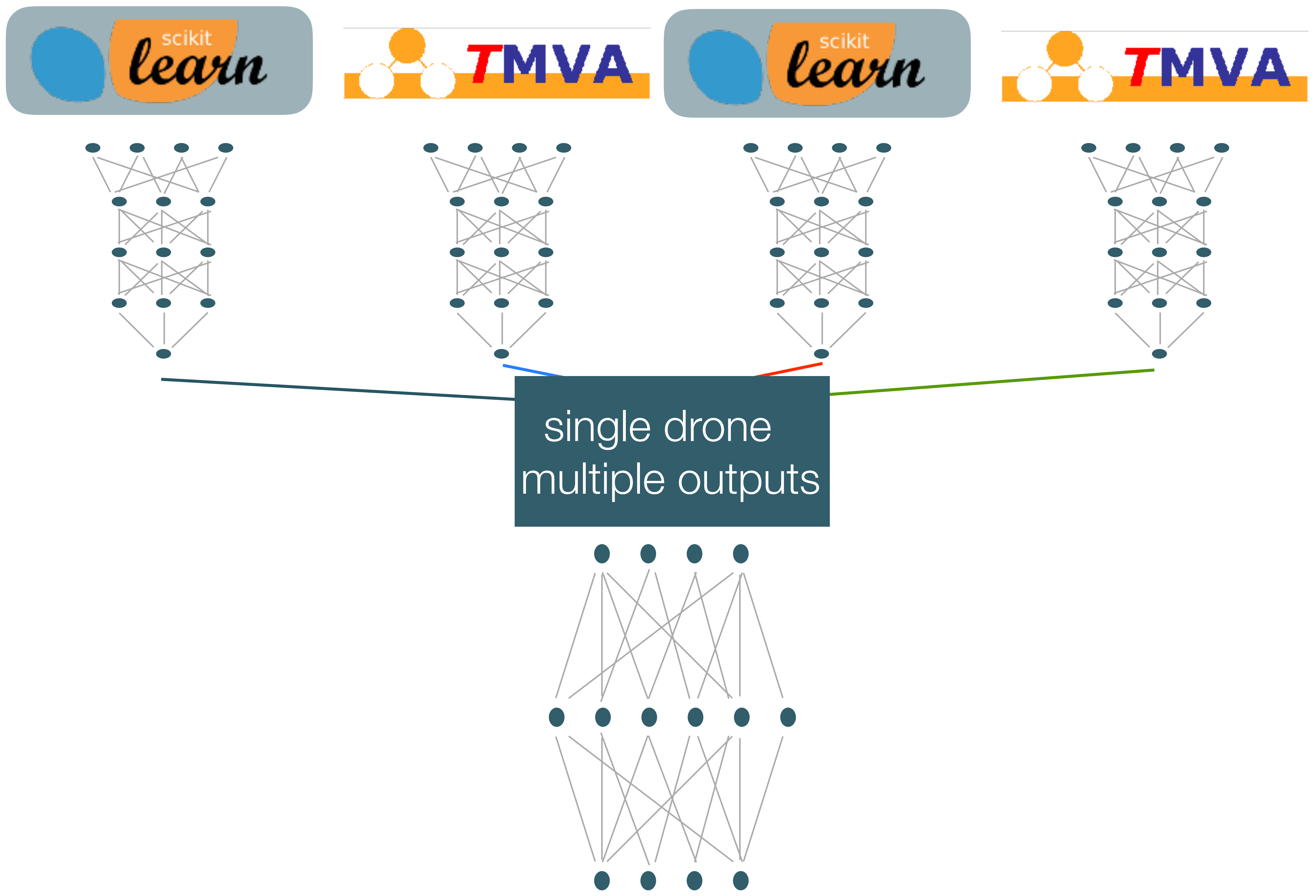
- Keras already provides the C++ to use in production environments
- LWTNN gives convenient saving and loading
- ONNX conversion makes life even easier.

Single algorithm, multiple classification, ubiquitous (dreaming)

Standard HEP filtering:

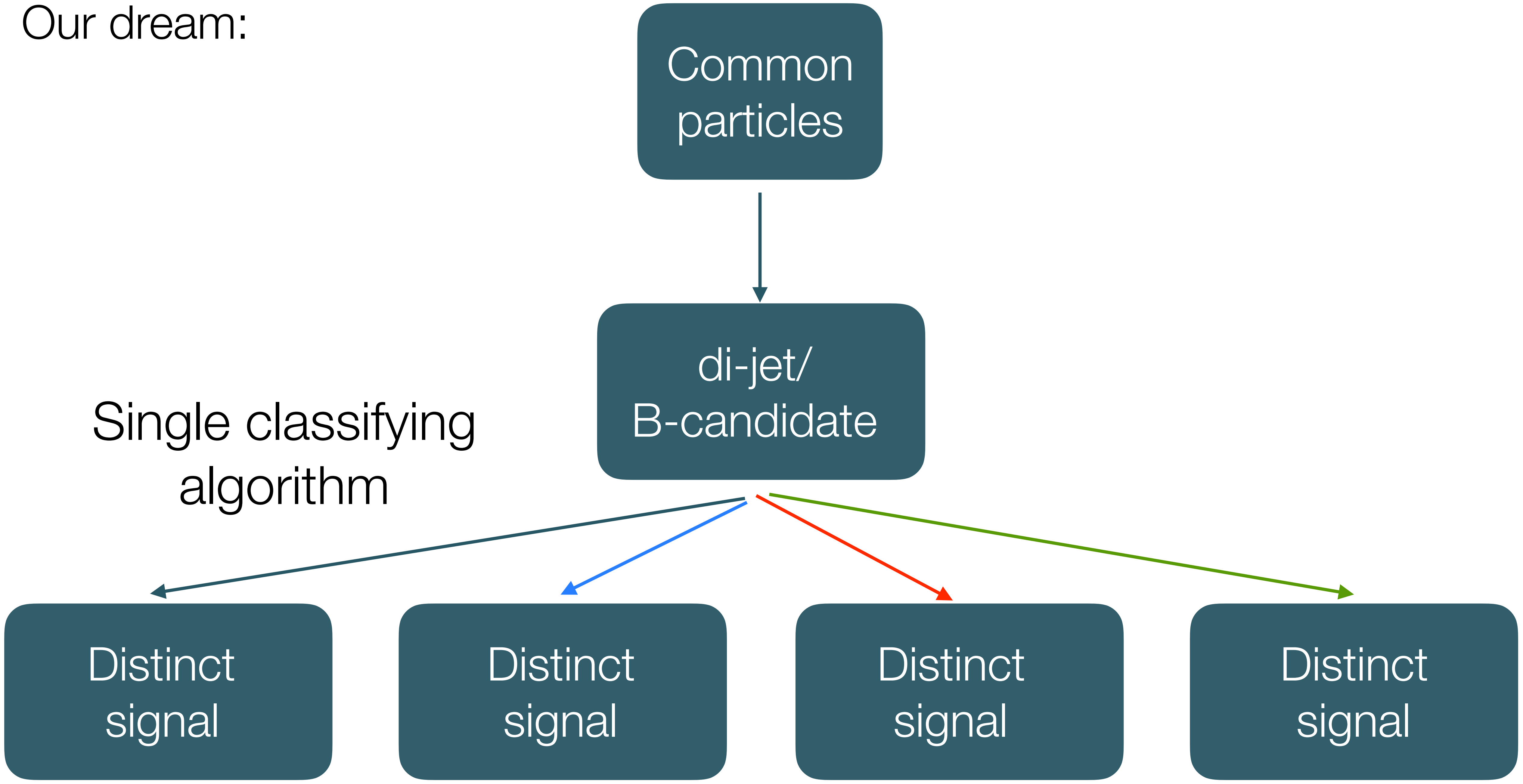


Single algorithm, multiple classification, ubiquitous (dreaming)



Single algorithm, multiple classification, ubiquitous (dreaming)

Our dream:



Repo/docs & final remarks

Networks presented here are available in the GitHub repo: HEPDrone, in addition to:

- Standard drone model
- Converter from original NN to drone
- JSON storage, for loading with LWTNN

Principle explained in: arXiv:1712.09114

Low latency environments face challenges which potentially restrict the freedom

We have demonstrated the approximation principle that allows our networks to learn the features of others with no access to the data

Such an extension can be used to not only speed-up the networks of different packages for use in low latency environments, but also makes use the ability of networks to provide more than one output classification.

- With an analyst able to provide a network for their particular signature of interest.